12 Сокеты и элементы сетевого программирования

12.1 Основы построения и функционирования вычислительных сетей (кратко)

Сеть – типичная ***сложная система*** (состоит из подсистем, которые, в свою очередь, также являются сложными системами).

Два основных уровня компонентов сети:
– базовая сеть передачи данных (СПД) – состоит, в свою очередь, из служебных узлов и каналов связи
– оконечные узлы (устройства пользователя, терминалы)

Два основных способа организации передачи данных в сетях: с установлением соединения и без установления соединения.

При передаче ***без установления соединения*** данные передаются в виде законченных самостоятельных блоков – ***датаграмм*** (***datagram***).

Каждая датаграмма доставляется к получателю по произвольному маршруту и независимо от других датаграмм, причем подтверждения о получении (квитирование) не предусматриваются, поэтому не гарантируется ни порядок следования датаграмм, ни единственность доставленного экземпляра, ни сам факт доставки – контролируются обычно только искажения каждой отдельной датаграммы. Просто, экономично, но для многих применений недостаточно надежно.

Передача ***с установлением соединения*** – обеспечение целостности и упорядоченности потока передаваемых данных. Независимо от способа организации потока данных порции данных доставляются получателю строго в том порядке, в котором они были отправлены, а прерывание потока своевременно распознается. Это достигается за счет нумерации порций данных и организации встречного потока подтверждений о получении (квитанций).

Таким образом образуется ***виртуальный канал*** передачи данных, для прикладных программ близкий по своим свойствам файлу или потоку ввода-вывода. Вместе с тем потоковая передача сложнее, требует специальных процедур установления соединения и дополнительных затрат на контроль его состояния, создает дополнительную нагрузку на линии связи в виде встречного потока квитанций.

Коммутация – распределения имеющихся физических каналов передачи данных для создания виртуальных.

Три основных метода коммутации:
– коммутация ***каналов***
– коммутация ***сообщений***
– коммутация ***пакетов***

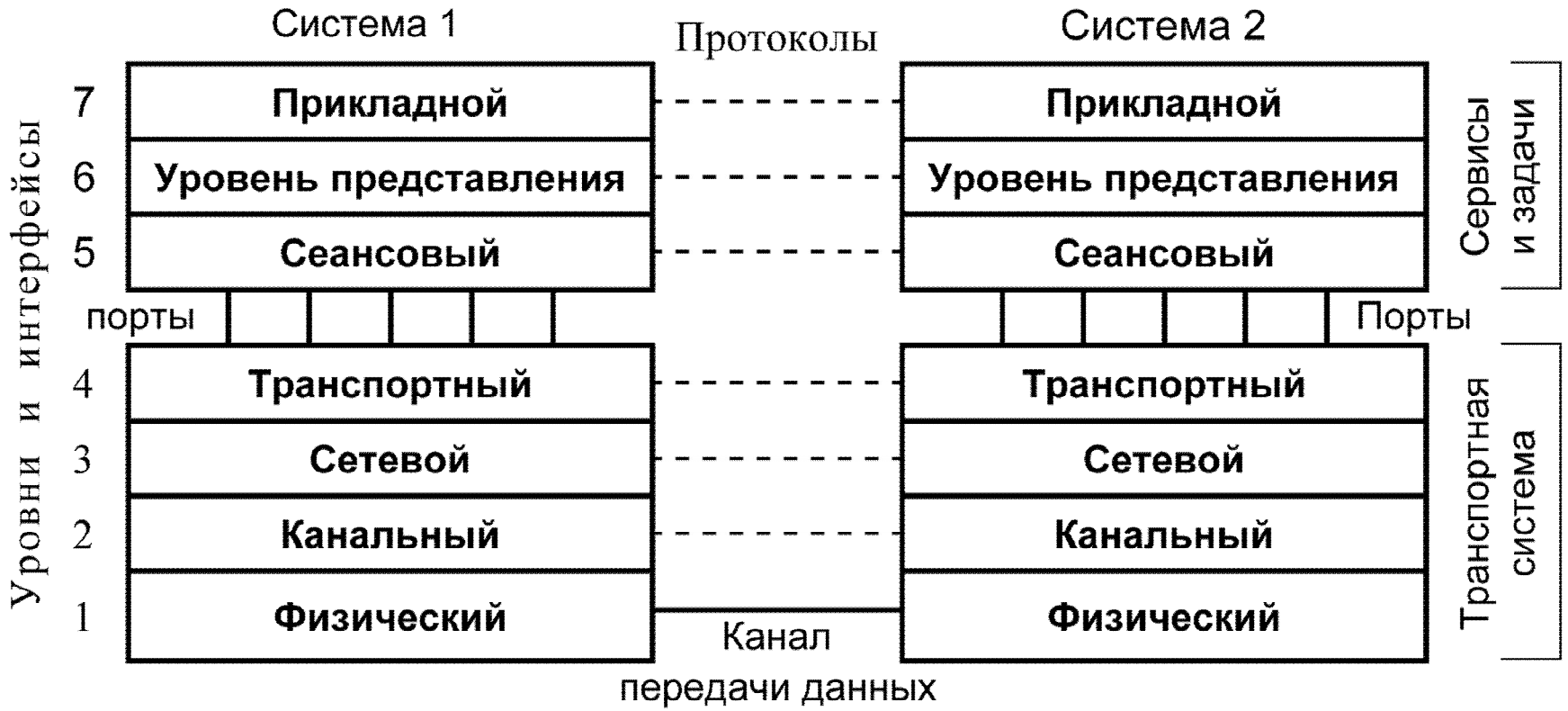
В современных сетях преобладает коммутация пакетов

Необходимость ***иерархического*** построения сетей и ***унификации*** в них.

Необходимость унификации построения разнородных систем и сетей и взаимодействия их друг с другом привела к переносу на них концепции открытых систем.

Открытая система – построена на основе открытых спецификаций.
Для сетей – открытость средств взаимодействия компонентов сети. Открытая система готова взаимодействовать с другими системами с использованием стандартных (унифицированных) правил.

Модель взаимодействия открытых систем – Open System Interconnection, ***OSI***
(принята Международной организацией по стандартизации (ISO) в 1983 г.)



Модель взаимодействия открытых систем для вычислительных сетей

Модификация модели для локальных вычислительных сетей

Протокол – набор правил и процедур взаимодействия между одноименными уровнями различных систем, обеспечивают корректную связь участников взаимодействия в сети.

Интерфейс – набор правил и средств их реализации для взаимодействия между соседними уровнями одной системы, обеспечивают возможность модульного построения системы.

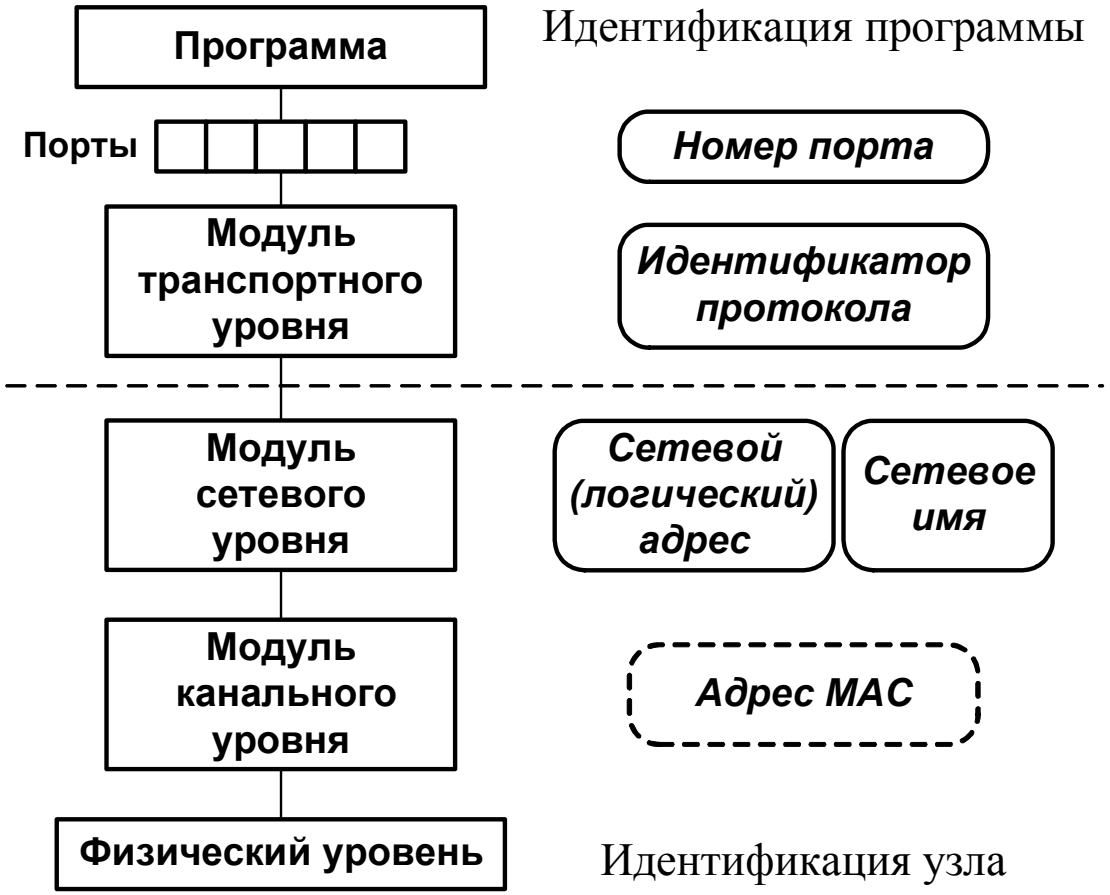
Стек протоколов в сети – набор протоколов, обслуживающих различные уровни взаимодействия. Протоколы в стеке проектируются с расчетом на совместную согласованную работу, но остаются достаточно независимыми для возможности замены на альтернативные с сохранением интерфейсов.

Далее будет рассматриваться в основном стек сетевых протоколов, обозначаемый как TCP/IP. В действительности он включает больше протоколов, относящихся к четырем уровням, но именно IP и TCP являются для него определяющими.

Инкапсуляция блоков данных (пакетов, сообщений) при прохождении через стек

Инкапсуляция блоков данных

Идентификация участников взаимодействия в сети – вписана в общую модель



Идентификация участников взаимодействия

В общем виде:

<сетевой адрес>:<протокол>:<порт>

(Здесь протокол может быть задан неявно – выбранным способом взаимодействия)

Разрешение адресов – преобразование адресов сетевого уровня в адреса канального уровня.

Сетевые ***имена***. ***Разрешение имен*** – преобразование сетевых имен в сетевые адреса.

Службы ***DNS***, ***NAT***

Сопоставление стеков TCP/IP и OSI

Центральное место – сетевой протокол ***IP*** и транспортные ***TCP*** и ***UDP***.

IP – ***I***nternet ***P***rotocol (version 4, 6) – сетевой уровень

UDP – ***U***ser ***D***atagram Protocol – транспортный уровень

TCP – ***T***ransmission ***C***ontrol ***P***rotocol – транспортный уровень

В силу особенностей протоколов и их реализаций, можно говорить об устойчивых связках протоколов – UDP+IP и особенно TCP+IP.

12.2 Программный интерфейс

Сокет (***socket***, букв. гнездо, соединитель) – программный объект, обычно системный, абстрагирующий точку доступа к транспортной системе.

В типичном случае сокет сопоставляется одному из ***портов*** одного из транспортных ***протоколов*** и служит удобной для прикладного программирования унифицированной надстройкой над ним. Реже используются сокеты, связанные с другими протоколами, либо «сырые» (неспецифицированные).

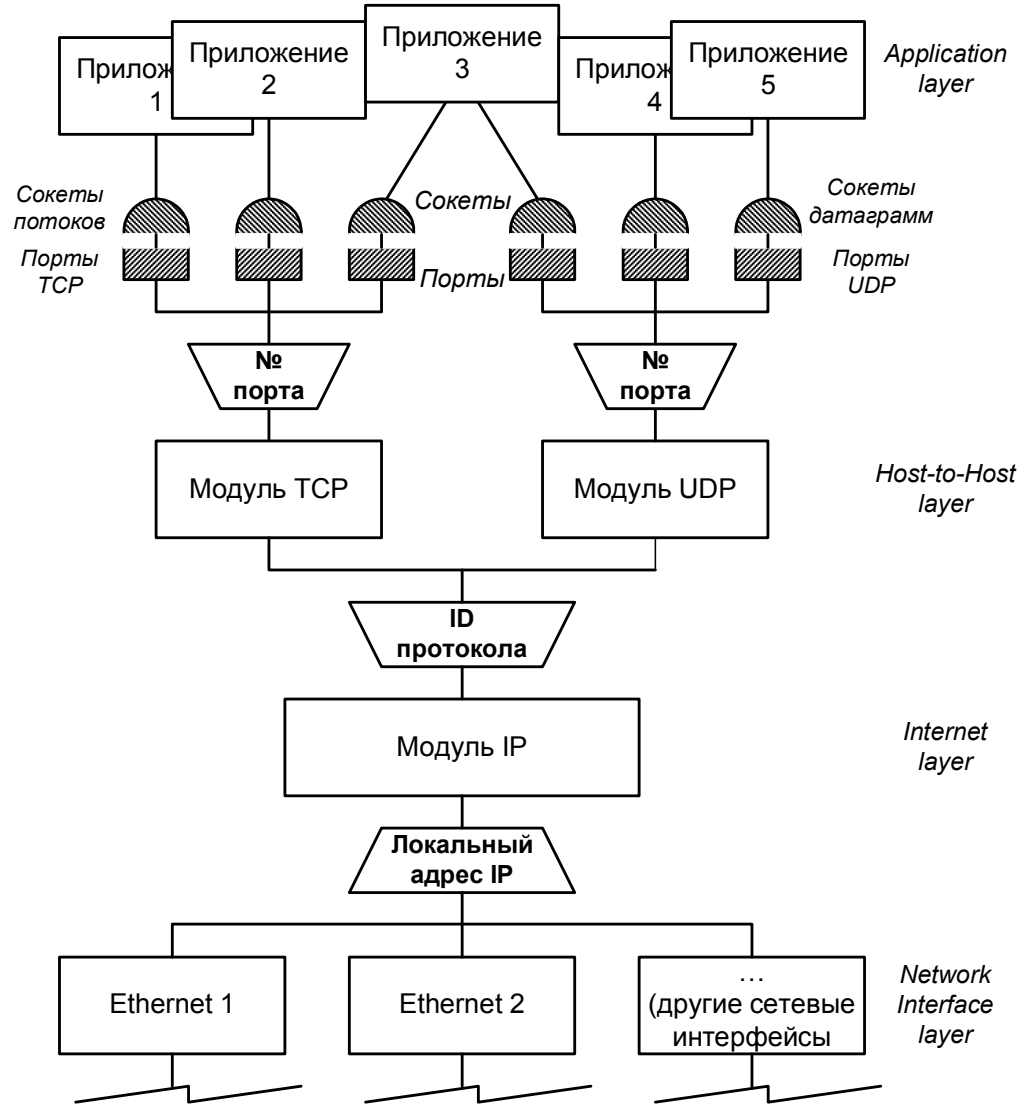
Программный интерфейс сокетов появился в BSD Unix и в дальнейшем стал фактическим стандартом для большинства систем. Функции работы с сокетами стандартизованы (входят в спецификации POSIX) и практически унифицированы между системами, для использующих их программ может быть обеспечена межплатформенная переносимость.

Как механизм взаимодействия, сокеты позволяют организовать его в том числе и между системами, в том числе на разных платформах (ОС, аппаратное обеспечение и проч.)

Тем не менее, возможны отличия в поведении некоторых функций, типах данных, именах заголовочных и библиотечных файлов т.п.

Реализация:
– Windows – подсистема Winsocket (Winsock, Winsock2), отдельный DLL и соответствующая библиотека (`.lib`)
– Unix – обычно реализация в ядре, библиотека поверх системных вызовов

Требования к компиляции и сборке в случае Linux:
– заголовочные файлы (основные): **`sys/socket.h`**, **`netinet/in.h`**, **`arpa/inet.h`**, **`netdb.h`**
– дополнительные опции **`gcc`** при сборке не требуются (для основной функциональности)



Мультиплексирование сообщений в стеке протоколов TCP/IP

Универсальность интерфейса сокетов требует достаточно сложного устройства с многочисленными настройками.

Характеристики сокета:

Тип («***семейств***о») ***протоколов***, «***домен***» (***domain***) – выбор стека протоколов (например, TCP/IP)

Тип («***семейств***о») ***адресации*** – выбор способа идентификации участников взаимодействия; обычно определяется стеком протоколов, но потенциально может различаться.

Адрес сокета – идентификатор сокета как точки взаимодействия, идентификатор использующего его процесса; формат адреса зависит от типа адресации

Тип сокета – неявно выбирает протокол взаимодействия в рамках семейства протоколов

Сокет в Unix-системах совместимы с обычными файловыми дескрипторами (**`fd`**) вплоть до возможности использовать их в функциях ввода-вывода (с последовательным доступом, подобно каналам).

Адрес сокета

Реализуется подобие «перегрузки» (в терминологии ООП) структуры адреса: «базовая структура» и ее «наследники». Передача структур как аргументов ***по указателю*** с ***приведением типа***.

«Базовая» структура адреса:

```
struct sockaddr {
  unsigned short sa_family; //семейство адресации
  char sa_data[14]; //резерв для данных адреса
};
```

И ее «наследник» – структура для TCP/IP («internet») адресов:

```
struct sockaddr_in {  //семейство адресации AF_INET
  short sin_family;  //= AF_INET
  unsigned short   sin_port; //номер порта
```

```
  struct in_addr    sin_addr; //значение IP-адреса
  char              sin_zero[8]; //заполнитель до 14
байт
};
```

```
struct in_addr {
  unsigned long s_addr; //значение IP_адреса как 32-
разрядное целое число
};
```

И IP-адрес, и номер порта должны быть записаны в «***сетевом***» формате, т.е. «***big endian***» (***MSB first***). Если они участвуют в вычислениях или в операциях ввода-вывода, необходимо выполнять преобразование между «сетевым» и «локальным» форматами (см. ниже).

Для сравнения – структура для «локальных» («Unix») адресов (имена в файловой системе):

```
struct sockaddr_un {  //семейство адресации AF_UNIX
  sa_family_t sun_family; // =AF_UNIX
  char        sun_path[108]; //имя файла и путь
};
```

Для заполнения адреса могут использоваться функции (неполный перечень):

```
inet_aton(), inet_pton()
inet_addr()
```

Обратное преобразование – из структуры адреса в строковую запись:

```
inet_ntoa(), inet_ntop()
```

Разрешение сетевого имени в адрес, фактически вызов связан с обращением к сетевой службе имен DNS:

`gethostbyname(), getnameinfo()`

Создание сокета

```
int socket(int domain, int type, int protocol);
```

Конфигурирование и подготовка сокета

Связывание сокета с локальным адресом (адресом сетевого интерфейса в локальной системе)

```
int bind(int sock, const struct sockaddr *addr,
socklen_t addrlen)
```

Связывание сокета с удаленным («серверным») сокетом

```
int connect(int sock, const struct sockaddr *addr,
socklen_t addrlen);
```

«Включение» сокета (для потоковых) с очередью ожидающих запросов:

```
int listen(int sock, int backlog);
```

Установка и получение параметров (опций) сокета, аргументы зависят от сокета и конкретных опций:

`setsockopt(…), getsockopt(…)`

Прием/передача данных

Прием соединений клиентов:

```
int accept(int sock, struct sockaddr *addr,
socklen_t *addrlen, int flags)
```

`accept()` – *прием соединения клиента*

Это особый случай приема данных: результатом является не собственно прикладные данные, а создание нового «клиентского» соединения. Обмен данными будет осуществляться через «клиентское» соединение, а сокет, на котором выполнялся accept(), остается доступным для новых соединений.

Прием и передача «прикладных» данных:

`recv(…), recvfrom(…)`
`send(…), sendto(…)`

Для «потоковых» сокетов возможно также использование обычных «файловых» функций **`read()`** и **`write()`**.

Закрытие сокета

Обычное закрытие аналогично файлу:

```
close( int sock)
```

Избирательное прекращение действия сокета на прием и/или передачу:

```
int shutdown(int sock, int how);
```

Низкоуровневое управление сокетом (по аналогии с устройствами и файлами):

`ioctlsocket(…)`

Прочие вспомогательные функции

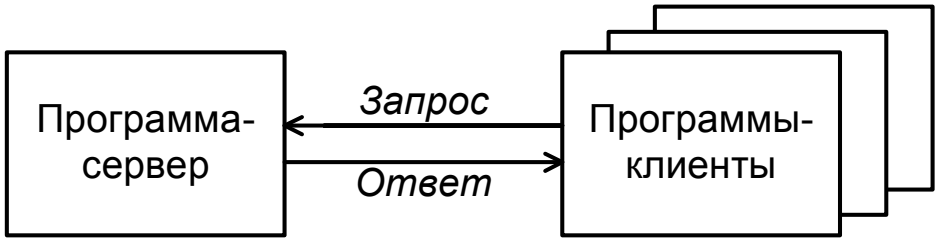
Согласование формата чисел (задача уровня представления в модели OSI) – локальный (***h***ost) и сетевой (***n***etwork) форматы для «коротких» (***s***hort) и «длинных» (***l***ong) целых чисел:

`htons(…), htonl(…), ntohs(…), ntohl()`

Стандартным сетевым считается формат «big endian», характерный для аппаратной архитектуры машин DEC. Архитектура Intel x86+ оперирует числами в формате «little endian» (младший байт по младшему адресу), поэтому преобразование необходимо.

12.3 Использование сокетов

Сетевые приложения обычно строятся в той или иной мере по схеме несимметричного взаимодействия «клиент-сервер»:
Сервер – программа, исполняющая поступающие запросы
Клиент – программа, отсылающая серверу запросы и принимающая результаты их обработки.



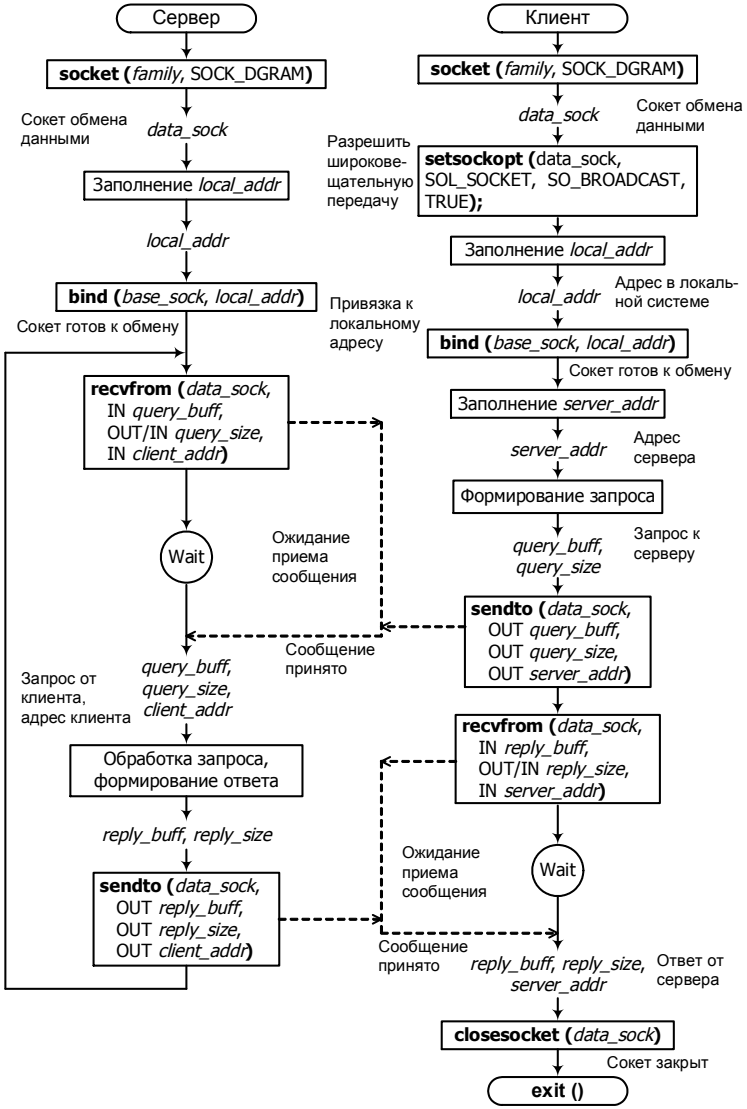
Взаимодействие «клиент-сервер»

11.3.1 Взаимодействие без установления соединения

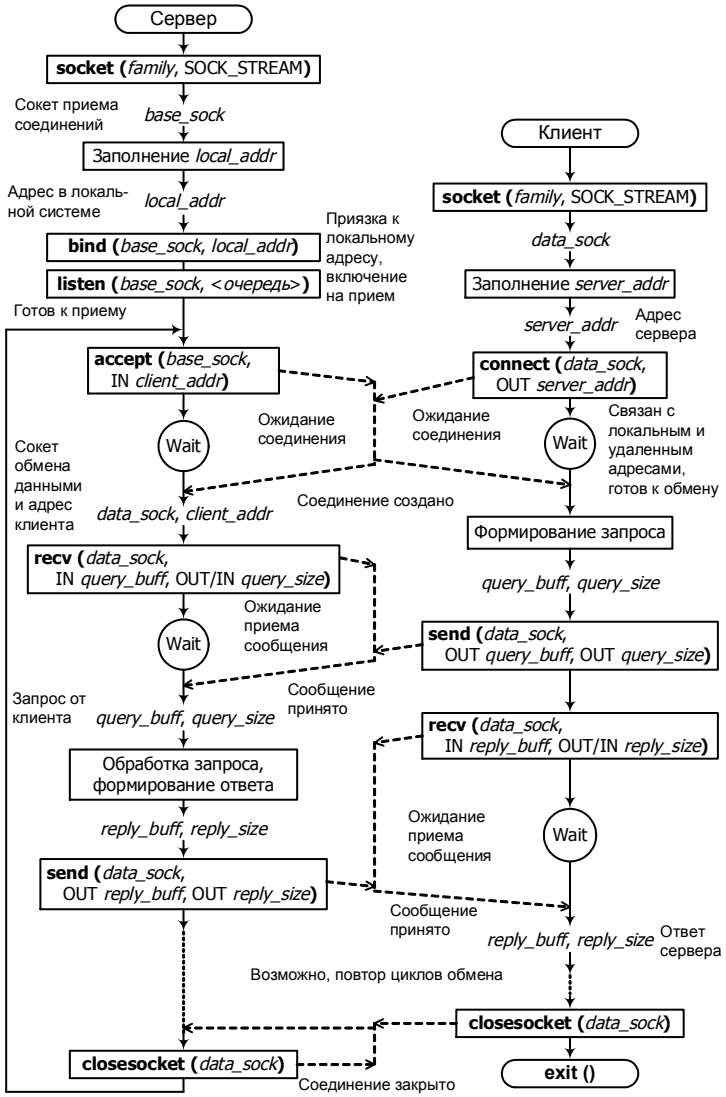
Взаимодействие потенциально симметрично, роли клиента и сервера различаются лишь логикой прикладного уровня.

11.3.2 Взаимодействие с установлением соединения

Взаимодействие принципиально несимметрично на уровне вызовов и использования сокетов.
Протокол TCP предусматривает отдельную «точку» (сокет) для приема запросов на соединение и одну или несколько для обмена данными. Встроенная в протокол обработка запроса на соединение (**`connect()`**)включает создание нового соединения между двумя новыми сокетами для обмена данными. Процедура может повторяться многократно, каждый раз создавая новое соединение. Закрытие соединения данных никак не сказывается на «базовой» точке.



Взаимодействие без установления соединения



Взаимодействие с установлением соединения

12.4 Построение многопользовательских серверов

Один сервер обычно может обслуживать множество клиентов, а клиент –обращаться сразу к нескольким серверам. Одна и та же программа может играть роль и клиента, и сервера, например, обращаясь к другому серверу в процессе исполнения запросов своих клиентов.

Обслуживание сервером нескольких клиентов требует в том или ином виде распараллеливания его работы. При использовании датаграммных сокетов каждое сообщение поступает и может быть обработано независимо от остальных, но при взаимодействии с установлением соединения необходимо специально заботиться о поддержании нескольких соединений.

Проблемы – начиная от параллельного приема новых соединений и обслуживания уже установленных.

12.4.1 Последовательный сервер

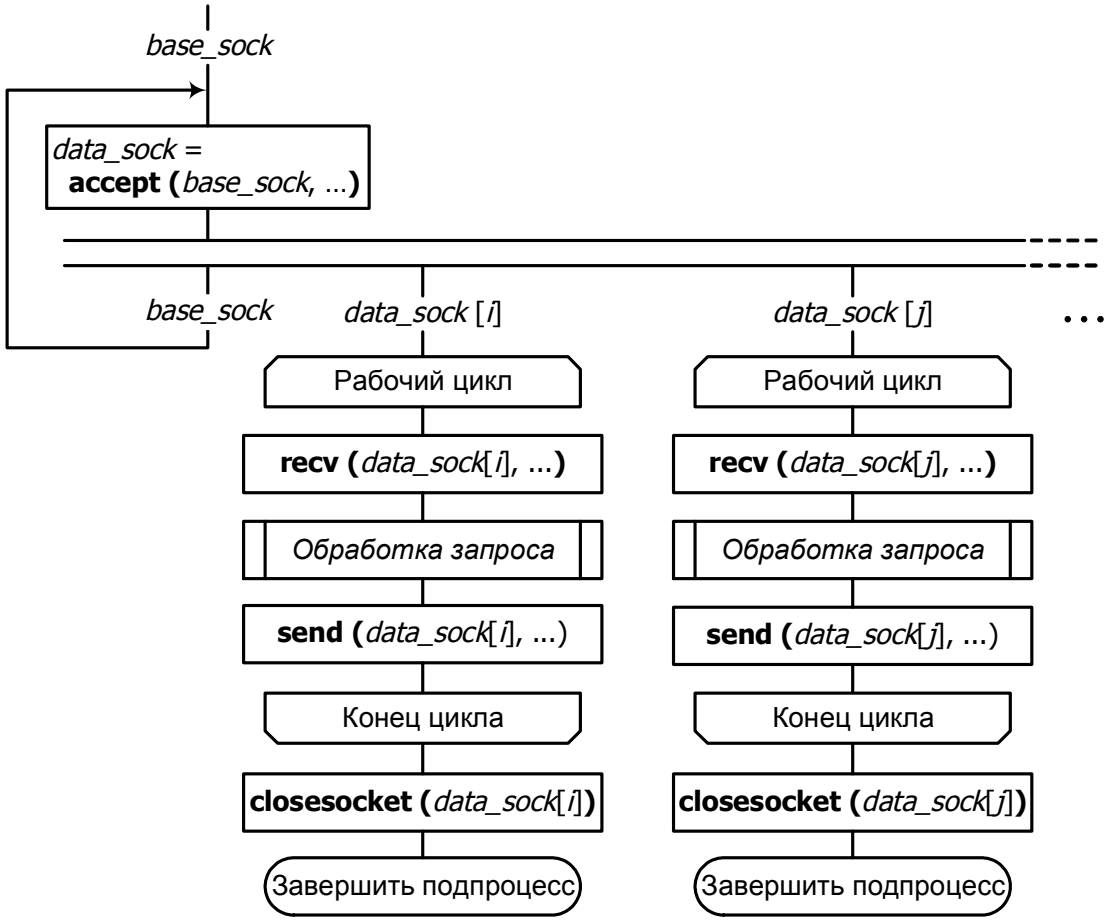
Простейший случай – последовательный сервер:
– прием запроса на соединение
– установление соединения
– обмен данными с клиентом
– закрытие соединения
– переход к следующим запросам на соединение

Минимальные затраты на проектирование и функционирование.

Фактически не предполагает параллельного исполнения запросов клиентов. Применим только в случае принципиально малой длительности и/или низкой интенсивности запросов.

Пример эффективного применения: сервер службы datetime.

12.4.2 Сервер с использованием многозадачности (многопоточности)



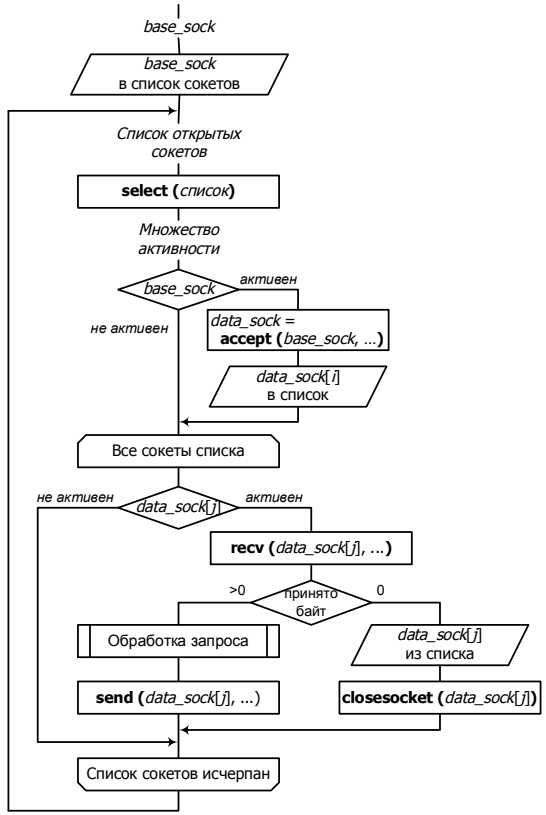
Многозадачный многопользовательский сервер

Каждое соединение, включая и ожидание новых запросов на соединение, обслуживается отдельным «подсервером», выполняющимся параллельно с другими.

Варианты:
– многозадачность на уровне ***процессов*** – надежность в случае аварийного поведения отдельных соединений, существенные затраты на процессы, затрудненное взаимодействие между «подсерверами»
– ***многопоточность*** – меньшие затраты, более эффективное взаимодействие между «подсерверами», меньшая (потенциально) устойчивость к сбоям в одном из потоков
– с ***предварительно созданными*** (***pre-created***, ***pre-forked***) потоками (процессами) – наличие ***пула*** потоков, откуда они выбираются для обслуживания очередного соединения, дополнительное избегание затрат на повторное создание потоков

В целом:
– логическая «прозрачность» архитектуры и удобство для проектирования
– хорошая масштабируемость
– дополнительные затраты на синхронизацию потоков (процессов) и типичные проблемы многозадачного программирования

12.4.3 Сервер с использованием мультиплексированного ввода-вывода (сервер на основе конечного автомата)



Сервер с мультиплексированием запросов

Предварительная проверка наличия данных (и готовности к передаче) всех открытых сокетов, прием и обработка имеющихся данных/запросов. Сеанс взаимодействия естественным образом разбивается на ряд состояний и переходов между ними. Сервер в заранее не известном порядке выполняет попеременно фрагменты алгоритмов, относящихся в разным соединениям.

В целом:
– логическая сложность проектирования (в том числе и для обеспечения надежности и отказоустойчивости)
– возможность реализации без поддержки многозадачности на уровне системы (актуально в прошлом)

Пример – посылка данных (структуры **TFrame**) через UDP

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <time.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>


struct TFrame { char cSignature[4];
 unsigned short wData1; unsigned long dwData2;
};
struct TFrame Frm;
```

```
int main (int argc, char** argv)
{
  struct sockaddr_in SockAddrLocal, SockAddrSend;
  int SockLocal = -1;
  struct hostent* pHostEnt;
  int nSockOptBC, nAddrSize, nPortRemote, nMsgLen;

//command line processing

  if (argc < 2) {
     fputs ("No enough arguments\n", stderr);
     fputs ("Use: UDPSend <addr/name> <port>\n",
stderr);
     return -1;
  }
```

```
//create local socket
 SockLocal = socket (PF_INET, SOCK_DGRAM, 0);
 if (SockLocal < 0) {
    fputs ("Error when creating socket\n", stderr);
    return -1;
 }
//configure socket: broadcasting is enabled
 nSockOptBC = 1;
 setsockopt (SockLocal, SOL_SOCKET,
    SO_BROADCAST, (char*) (&nSockOptBC),
sizeof(nSockOptBC));
//bind socket to "local" address
 memset (&SockAddrLocal, 0, sizeof(SockAddrLocal));
 SockAddrLocal.sin_family = AF_INET;
 SockAddrLocal.sin_addr.s_addr = INADDR_ANY; //all
```

```
  SockAddrLocal.sin_port = 0; //"auto" port number
  if (bind(SockLocal,(struct
sockaddr*)&SockAddrLocal,
       sizeof(SockAddrLocal)) != 0)
  {
     fputs ("Error when binding socket to local
address\n", stderr);
     return -1;
  }
//prepare "remote" address
  memset (&SockAddrSend, 0, sizeof (SockAddrSend));
  SockAddrSend.sin_family = AF_INET;
  if (strcmp (argv[1], "255.255.255.255") == 0)
//адрес broadcast
     SockAddrSend.sin_addr.s_addr= INADDR_BROADCAST;
  else {
```

```
    SockAddrSend.sin_addr.s_addr = inet_addr
(argv[1]);
    if (SockAddrSend.sin_addr.s_addr ==
INADDR_NONE) {
       if ((pHostEnt = gethostbyname (argv[1])) ==
NULL) {
          fprintf (stderr, "Wrong or unrecognized
host: %s\n",
             argv[1]);
          return -1;
       }
       SockAddrSend.sin_addr =
          *(struct in_addr*)(pHostEnt-
>h_addr_list[0]);
    }
 }
```

```
  if (sscanf (argv[2], "%u", &nPortRemote) < 1) {
     fprintf (stderr, "Wrong port number: %s\n",
argv[2]);
     return -1;
  }
  SockAddrSend.sin_port = htons ((unsigned
short)nPortRemote);

//initialize data generation

  srand( time( NULL));
  fprintf( stderr, "Sizeof \"Frame\" structure:
%u\n", sizeof(TFrame));

//main function loop

  while (1)
  {
  //data to be sent
```

```
    Frm.cSignature[0] = 'F'; Frm.cSignature[1] =
'R';
    Frm.cSignature[2] = 'M'; Frm.cSignature[3] =
'\0';
    Frm.wData1 = rand();
    Frm.dwData2 = (rand() << 16) | rand();
```

```
  //send message
     fprintf( stdout, "Send to %s:%u: { \"%s\";
%04X; %08X }\n",
     inet_ntoa(SockAddrSend.sin_addr),
ntohs(SockAddrSend.sin_port),
     &(Frm.cSignature[0]),
     (unsigned)Frm.wData1, Frm.dwData2);
     nMsgLen = sizeof(Frm);
     if (sendto (SockLocal, &Frm, nMsgLen, 0,
     (struct sockaddr*) &SockAddrSend, sizeof
(SockAddrSend)) < nMsgLen)
     {
        fprintf (stderr, "Error when sending:
\"%s\"\n",
           &(Frm.cSignature[0]));
        continue;
```

```
      }
      sleep( 1);
   }
  //close all
   fputs( "Sender was stopped by user\n", stderr);
   shutdown (SockLocal, 2);
   sleep (1);
   close(SockLocal); SockLocal = -1;
   return 0;
}
```